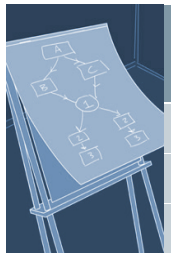
Chapter 2.
Combinational Logic Circuits

Part II. Circuit Optimization

Mar., 2008



4. 2단계 회로 최적화

5. Map Manipulation
6. Practival Optimization (Espresso)
7. Multi-level Circuit Optimization

Overview

- Part 1 – Gate Circuits and Boolean Equations
  - **Binary Logic and Gates**
  - **Boolean Algebra**
  - **Standard Forms**
- Part 2 – Circuit Optimization
  - **Two-Level Optimization**
  - **Map Manipulation**
  - **Practical Optimization (Espresso)**
  - **Multi-Level Circuit Optimization**
- Part 3 – Additional Gates and Circuits
  - **Other Gate Types**
  - **Exclusive-OR Operator and Gates**
  - **High-Impedance Outputs**

Circuit Optimization

- Goal: To obtain the simplest implementation for a given function
- Optimization is a more formal approach to simplification that is performed using a specific procedure or algorithm

  (In computing, *optimization* is the process of modifying a system to make some aspect of it work more efficiently or use fewer resources. For instance, a computer program may be optimized so that it executes more rapidly, or is capable of operating with less memory storage or other resources, or draw less power)
- Optimization requires a cost criterion to measure the simplicity of a circuit
- Distinct cost criteria we will use:
  - **Literal cost (L)**
  - **Gate input cost (G)**
  - **Gate input cost with NOTs (GN)**

Literal Cost

- Literal – a variable or its complement
- Literal Cost (문자비용) – the number of literal appearances in a Boolean expression corresponding to the logic circuit diagram
- Examples:
  - $F = BD + A\bar{B}C + AC\bar{D}$ L = 8
  - $F = BD + A\bar{B}C + A\bar{B}\bar{D} + AB\bar{C}$ L =
  - $F = (A + B)(A + D)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$ L =
  - **Which solution is best?**

Gate Input Cost

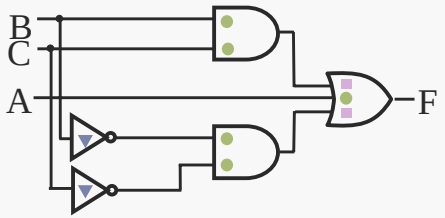
- Gate input cost (게이트입력비용) - the number of inputs to the gates in the implementation corresponding exactly to the given equation or equations. (G - inverters not counted, GN - inverters counted)
- For SOP and POS equations, it can be found from the equation(s) by finding the sum of:
  - **all literal appearances**
    (나타나는 모든 문자수)
  - **the number of terms excluding single literal terms (G) and**
    (오직 하나의 문자로 구성된 항들을 제외한 항들의 수)
  - **optionally, the number of distinct complemented single literals (GN)**
    (선택적으로 유일하게 반전된 단일 문자들의 수)

- Example: (Page 73)
  - $G = A\,B\,C\,D + \bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$ L=8, G=10, GN=14
  - $G' = (\bar{A} + B)(\bar{B} + C)(\bar{C} + D)(\bar{D} + A)$ L=8, G=12, GN=16
    - **문자비용(Literal Cost)은 같지만, 게이트입력비용(Gate Input Cost)에서 G가 G'보다 낮은 값을 갖는다.**
- 게이트 입력비용은 논리회로의 구현에 사용되는 트랜지스터와 전선의 수에 비례하는 수치이므로 현재의 논리 구현에서의 좋은 측정 기준이 된다.
- Example:
  - $F = BD + A\bar{B}C + AC\bar{D}$ G = 8, GN = 11
  - $F = BD + A\bar{B}C + A\bar{B}\bar{D} + AB\bar{C}$ G = , GN =
  - $F = (A + \bar{B})(A + D)(B + C + \bar{D})(\bar{B} + \bar{C} + D)$ G = , GN =
  - **Which solution is best?**

Cost Criteria (continued)

- Example 1:
- $F = A + B\,C + \bar{B}\,\bar{C}$

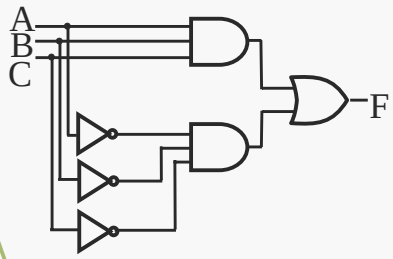
GN = G + 2 = 9
L = 5
G = L + 2 = 7



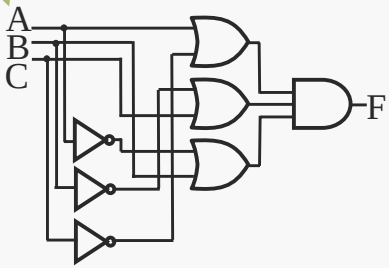
- **L (Literal Count) counts the AND inputs and the single literal OR input**
- **G (Gate Input Count) adds the remaining OR gate inputs**
- **GN (Gate Input Count with NOTs) adds the inverter inputs**

Cost Criteria (continued)

- Example 2:
- **F = A B C + $\bar{A}$ $\bar{B}$ $\bar{C}$**
- L=6, G=8, GN=11



- **F' = (A + $\bar{C}$)($\bar{B}$ + C)($\bar{A}$ + B)**
- L=6, G=9, GN=12



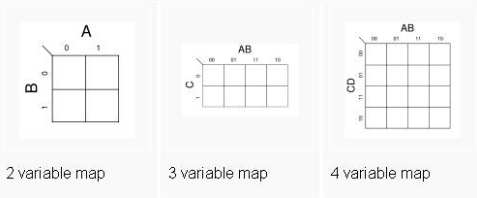
- Same function and Same literal cost
- But first circuit (F) has better gate input count and better gate input count with NOTs
- Select it!

Boolean Function Optimization

- Minimizing the gate input (or literal) cost of a (a set of) Boolean equation(s) reduces circuit cost.
- We choose gate input cost.
- Boolean Algebra and graphical techniques are tools to minimize cost criteria values.
- Some important questions:
  - **When do we stop trying to reduce the cost?**
  - **Do we know when we have a minimum cost?**
- Treat optimum or near-optimum cost functions for two-level (SOP and POS) circuits first.
- Introduce a graphical technique using Karnaugh maps (K-maps, for short)

Karnaugh Map

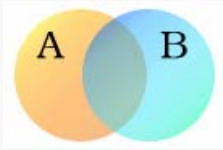
- The **Karnaugh map**, also known as a **Veitch diagram** (K-map or KV-map for short), is a tool to facilitate management of Boolean algebraic expressions. A Karnaugh map is unique in that only one variable changes value between squares; in other words, the rows and columns are ordered according to the principles of Gray code.
- Size of Map
  - In a Karnaugh map with *n* variables, a Boolean term mentioning *k* of them will have a corresponding rectangle of area 2^{n-k}. Common sized maps are of 2 variables which is a 2x2 map; 3 variables which is a 2x4 map; and 4 variables which is a 4x4 map.



Karnaugh Maps (K-map)

- A K-map is a collection of squares
  - **Each square represents a minterm**
  - **The collection of squares is a graphical representation of a Boolean function** (Any Boolean function can be expressed as s Sum of Minterms)
  - **Adjacent squares differ in the value of one variable**
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares
  - 진리표를 그레이코드로 나타낸 변수들의 조합을 2진법 순서로 나열하여 도표로 간략화시킨다
- The K-map can be viewed as
  - A reorganized version of the truth table
  - A topologically-warped Venn diagram as used to visualize sets in algebra of sets (벤다이어그램을 확장한 것으로 볼 수 있다)

- **Venn Diagram**
  - are illustrations used in the branch of mathematics known as set theory. They show all of the possible mathematical or logical relationships between sets (groups of things). They normally consist of overlapping circles.
  - 서로 다른 집합들 사이의 관계를 보여주기 위한 그림
  - For instance, in a two-set Venn diagram, one circle may represent the group of all wooden objects, while another circle may represent the set of all tables.



Some Uses of K-Maps

- Provide a means for:
  - **Finding optimum or near optimum**
    - **SOP and POS standard forms, and**
    - **two-level AND/OR and OR/AND circuit implementations**

    for functions with small numbers of variables
  - **Visualizing concepts related to manipulating Boolean expressions, and**
  - **Demonstrating concepts used by computer-aided design programs to simplify large circuits**

Two Variable Maps

- A 2-variable Karnaugh Map:
  - Note that minterm m_0 and minterm m_1 are "*adjacent*" and differ in the value of the variable y
  - Similarly, minterm m_0 and minterm m_2 differ in the x variable.
  - Also, m_1 and m_3 differ in the x variable as well.
  - Finally, m_2 and m_3 differ in the value of the variable y

	y=0	***y*=1**
***x*=0**	$m_0 = \bar{x}\bar{y}$	$m_1 = \bar{x}y$
***x*=1**	$m_2 = x\bar{y}$	$m_3 = xy$

K-Map and Truth Tables

- The K-Map is just a different form of the truth table.
- Example – Two variable function:
  - We choose a,b,c and d from the set {0,1} to implement a particular function, F(x,y)

Function Table

Input Values (x,y)	**Function Value F(x,y)**
0 0	**a**
0 1	**b**
1 0	**c**
1 1	**d**

K-Map

	y = 0	**y = 1**
x = 0	**a**	**b**
x = 1	**c**	**d**

K-Map Function Representation

- Example: **F(x,y) = x**

F = x	y = 0	y = 1
x = 0	0	0
x = 1	1	1

- For function F(x,y), the two adjacent cells containing 1's can be combined using the Minimization Theorem:

$$F(x,y) = x\bar{y} + xy = x$$

K-Map Function Representation

- Example: **G(x,y) = x + y**

G = x+y	y = 0	y = 1
x = 0	0	1
x = 1	1	1

- For G(x,y), two pairs of adjacent cells containing 1's can be combined using the Minimization Theorem:

$$G(x,y) = (x\bar{y} + xy) + (xy + \bar{x}y) = x + y$$

Duplicate x y

2-variable maps

1. Enter the function on the K-map
   (K맵에 함수를 표시한다)
2. Identify collections of squares on the map representing product terms to be considered for the simplified expression (*rectangles*)
   (간략화된 표현식을 위하여 고려해야할 곱항을 표시하는 사각형들을 수집한다)
   - Goal is to find the fewest such rectangles (contain numbers of squares that are powers of 2) that include or cover all of the wquares marked with 1s
3. Determine if any of the rectangles we have generated is not needed to cover all of the 1s on the K-map
   (구해진 사각형들에서 K맵에서의 모든 1을 커버하기 위해 필요하지 않은 사각형들을 결정한다)
4. Read off the sum-of-products expression, determining the corresponding product terms for the required rectangles in the map
   (맵에서 남아있는 사각형들로부터 해당 곱항들을 구하여 곱합의 합 표현식을 구한다)

Page77

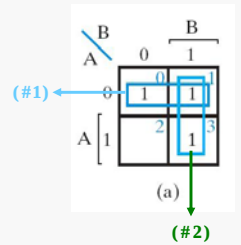
- Given F(x,y) as Table 2-9,

(1) F가 1을 갖는 각 행에 대하여 맵상에 1을 표시하고 0을 표시하지 않는다
F(A,B) = $\sum m(0,1,3)$ → K맵에서 0,1,3에 해당하는 위치에 1을 입력

SOP로 주어질 때,
$F = \bar{A} + AB$

TABLE 2-9
Two-Variable Function F(A, B)

A	B	F
0	0	1
0	1	1
1	0	0
1	1	1



(2) 1번 사각형에 해당는 minterm은 $\bar{A}\bar{B} + \bar{A}B = \bar{A}$
2번 사각형에 해당는 minterm은 $AB + \bar{A}B = B$

Finally, $F = \bar{A} + B$

Three Variable Maps

- A three-variable K-map:

	yz=00	yz=01	yz=11	yz=10
x=0	m_0	m_1	m_3	m_2
x=1	m_4	m_5	m_7	m_6

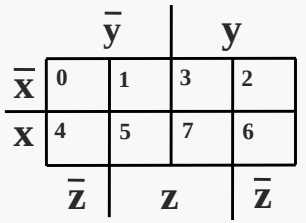
- where each minterm corresponds to the product terms:

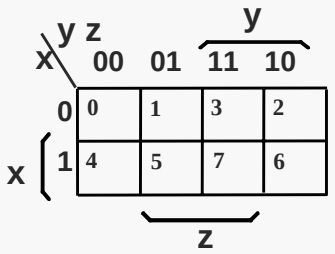
	yz=00	yz=01	yz=11	yz=10
x=0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
x=1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

- Note that if the binary value for an index differs in one bit position, the minterms are adjacent on the K-Map

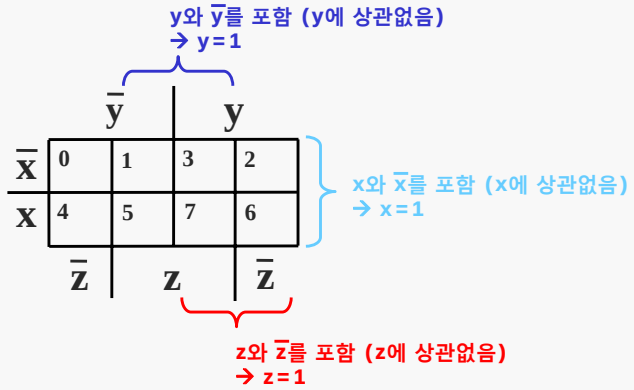
Alternative Map Labeling

- Map use largely involves:
  - **Entering values into the map, and**
  - **Reading off product terms from the map.**
- Alternate labelings are useful:





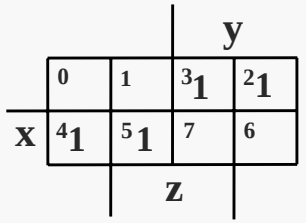
- **Read off**



Example Functions

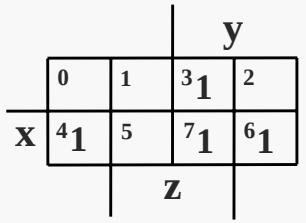
- By convention, we represent the minterms of F by a "1" in the map and leave the minterms of $\bar{F}$ blank
- Example:

$$F(x, y, z) = \Sigma_m(2,3,4,5)$$



- Example:

$$G(a, b, c) = \Sigma_m(3,4,6,7)$$



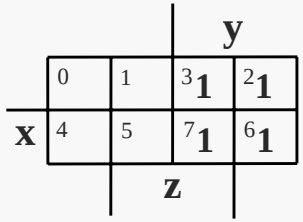
- Learn the locations of the 8 indices based on the variable order shown (x, most significant and z, least significant) on the map boundaries

Combining Squares (*rectangles*)

- By combining squares, we reduce number of literals in a product term, reducing the literal cost, thereby reducing the other two cost criteria
- On a 3-variable K-Map:
  - **One square represents a minterm with three variables**
  - **Two adjacent squares represent a product term with two variables** (1*2, 2*1)
  - **Four "adjacent" terms represent a product term with one variable** (1*4, 4*1, 2*2)
  - **Eight "adjacent" terms is the function of all ones (no variables) = 1** (2*4)

Example: Combining Squares

- Example: Let $F = \Sigma m(2,3,6,7)$



- Applying the Minimization Theorem three times:

$$F(x,y,z) = \bar{x}yz + xyz + \bar{x}y\bar{z} + xy\bar{z}$$
$$= yz + y\bar{z}$$
$$= y$$

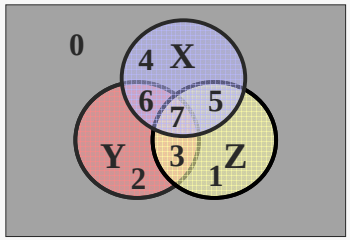
- Thus the four terms that form a 2 × 2 square correspond to the term "y".

Three-Variable Maps

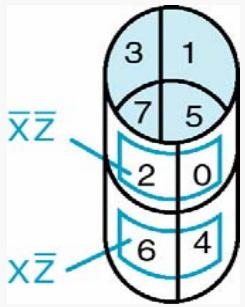
- Reduced literal product terms for SOP standard forms correspond to rectangles on K-maps containing cell counts that are powers of 2.
- Rectangles of 2 cells represent 2 adjacent minterms; of 4 cells represent 4 minterms that form a "pairwise adjacent" ring.
- Rectangles can contain non-adjacent cells as illustrated by the "pairwise adjacent" ring above.

Three-Variable Maps

- Topological warps of 3-variable K-maps that show *all* adjacencies:
  - **Venn Diagram**

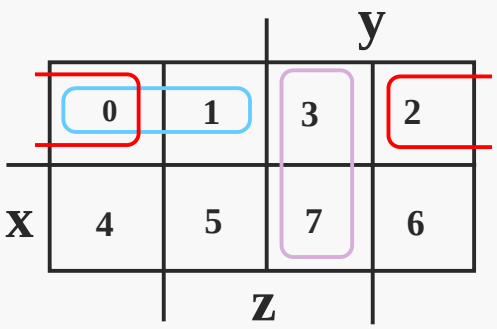


  - **Cylinder**



Three-Variable Maps

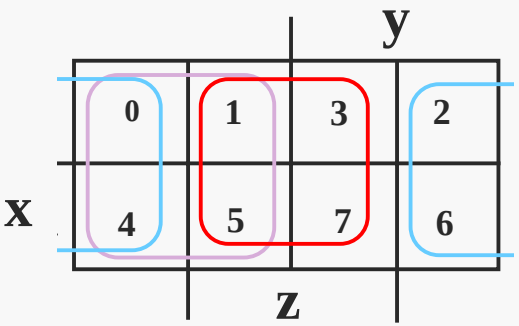
- Example Shapes of 2-cell Rectangles:



- Read off the product terms for the rectangles shown

Three-Variable Maps

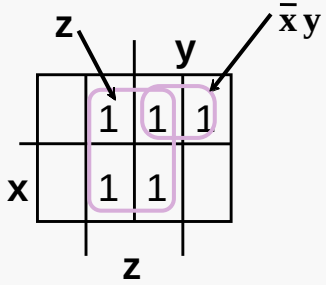
- Example Shapes of 4-cell Rectangles:



- Read off the product terms for the rectangles shown

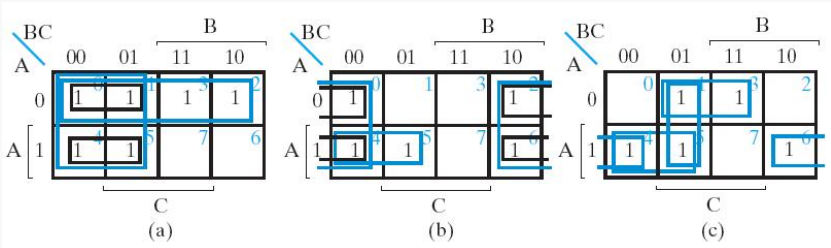
Three Variable Maps

- K-Maps can be used to simplify Boolean functions by systematic methods. Terms are selected to cover the "1s"in the map.
- Example: **Simplify** $F(x, y, z) = \Sigma_m(1,2,3,5,7)$



$$F(x, y, z) = z + \bar{x}\, y$$

- **Page 79~81, Example 2-5,6,7**
  - **F(A,B,C) = ∑ *m*(0,1,2,3,4,5) → simplified F is ??**
  - **G(A,B,C) = ∑ *m*(0,2,4,5,6) → G ??**
  - **H(A,B,C) = ∑ *m*(1,3,4,5,6) → H ??**



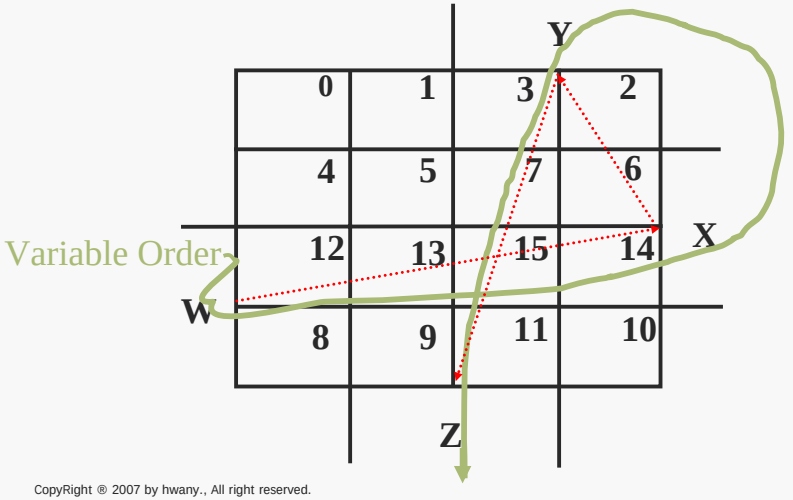
Three-Variable Map Simplification

- Use a K-map to find an optimum SOP equation for

$$F(X, Y, Z) = \Sigma_m(0,1,2,4,6,7)$$

Four Variable Maps

- Map and location of minterms:

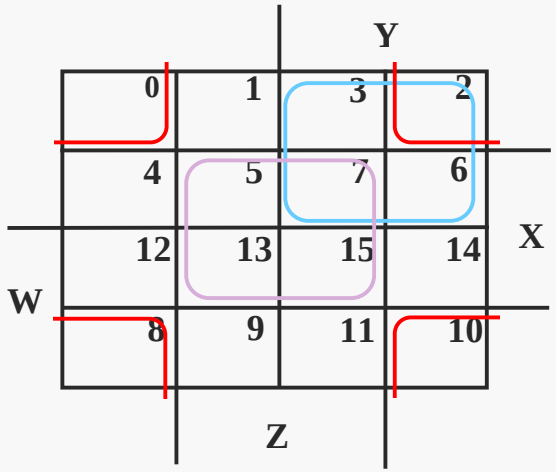


Four Variable Terms

- Four variable maps can have rectangles corresponding to:
  - A single 1 = 4 variables, (i.e. Minterm)
  - Two 1s = 3 variables,
  - Four 1s = 2 variables
  - Eight 1s = 1 variable,
  - Sixteen 1s = zero variables (i.e. Constant "1")

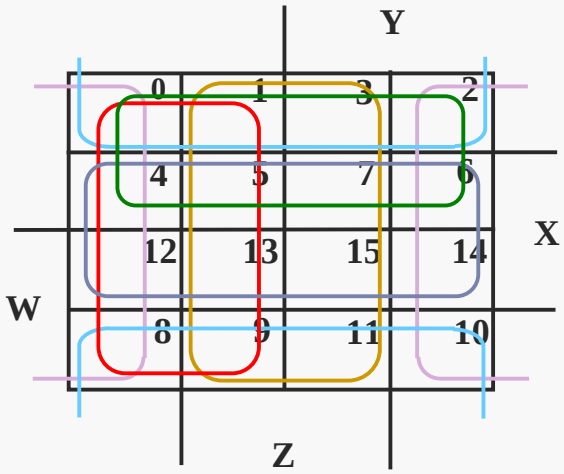
Four-Variable Maps

- Example Shapes of Rectangles:



Four-Variable Maps

- Example Shapes of Rectangles:



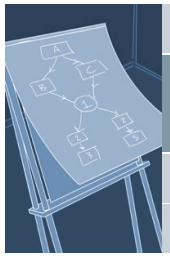
- **Page 81~83, Example 2-8,9**
  - **F(A,B,C,D) = ∑ *m*(0,1,2,4,5,6,8,9,10,12,13) → F is ??**
  - **G(A,B,C,D) = $\overline{A}\,\overline{C}\,\overline{D} + \overline{A}\,D + \overline{B}\,C + C\,D + A\,\overline{B}\,\overline{D}$ → G ??**

Four-Variable Map Simplification

$$F(W, X, Y, Z) = \Sigma_m(0, 2,4,5,6,7,8,10,13,15)$$

Four-Variable Map Simplification

$$F(W, X, Y, Z) = \Sigma_m(3,4,5,7,9,13,14,15)$$



4. 2-level Circuit Optimization

5. 맵 조작

6. Practival Optimization (Espresso)

7. Multi-level Circuit Optimization

Systematic Simplification

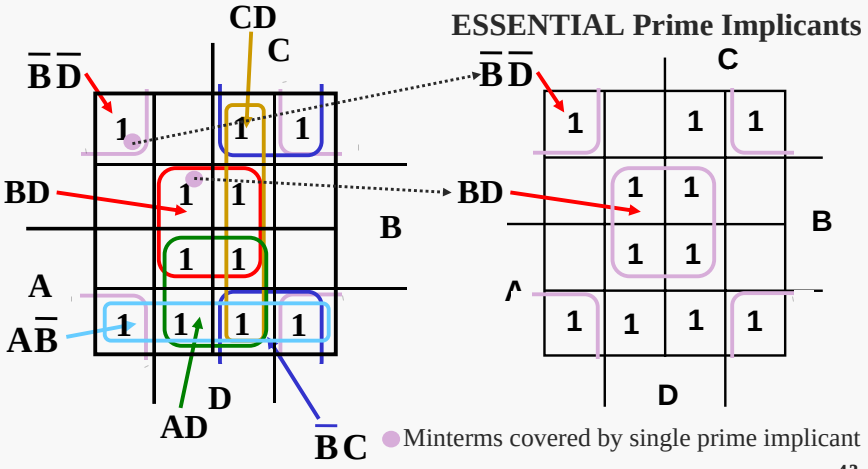
- A *Prime Implicant* (주항) is a product term obtained by combining the maximum possible number of adjacent squares in the map into a rectangle with the number of squares a power of 2.

  (맵에서 1을 포함한 정사각형들이 사각형의 형태로 최대한 크게 2^m개씩 조합하여 구할 수 있다)

- A prime implicant is called an *Essential Prime Implicant* (필수주항) if it is the only prime implicant that covers (includes) one or more minterms.
- Prime Implicants and Essential Prime Implicants can be determined by inspection of a K-Map.
- A set of prime implicants *"covers all minterms"* if, for each minterm of the function, at least one prime implicant in the set of prime implicants includes the minterm.

Example of Prime Implicants

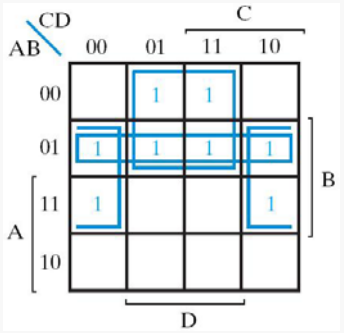
- Find ALL Prime Implicants



Example of Prime Implicants

- Page 84, Example 2-10

  Find all prime implicants in Figure 2-13,
  and then, write the simplified expression for function F



  - **Also, see example 2-11 on page 84**

- Identification of Essential Prime Implicants in the map
  - provides an additional tool which shows the terms that must absolutely appear in every sum-of-products expression for a function

    (함수에 대한 개개의 합의 항 표현에서 반드시 나타나야 하는 항을 발견하는 방법을 제시)
  - provides a partial structure for a more systematic method for choosing patterns of prime implicants

    (정사각형의 형태를 선택하는 것에서 더욱 규칙적인 방법을 위한 중요한 부분에서의 방법을 제공)

Prime Implicant Practice

- Find all prime implicants for:

$$F(A,B,C,D) = \Sigma_m(0,2,3,8,9,10,11,12,13,14,15)$$

Another Example

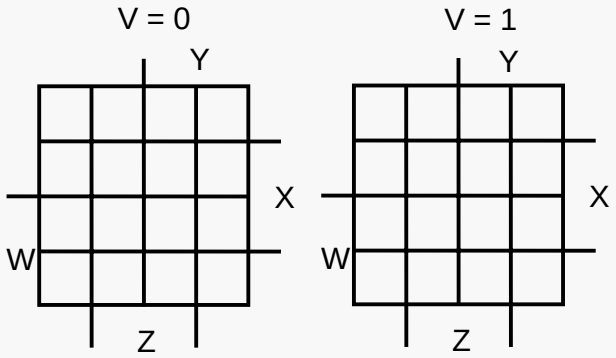
- Find all prime implicants for:

$$G(A,B,C,D) = \Sigma_m(0,2,3,4,7,12,13,14,15)$$

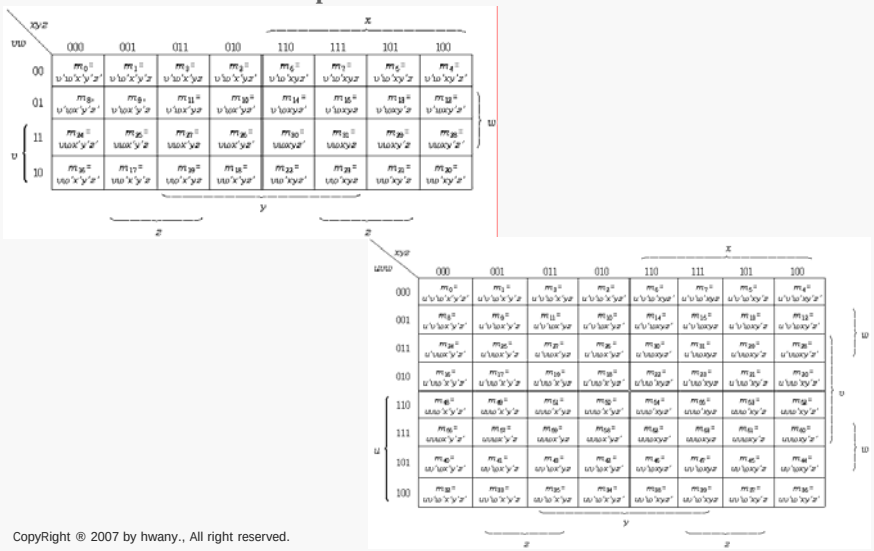
  - Hint: There are seven prime implicants!

Five Variable or More K-Maps

- For five variable problems, we use *two adjacent K-maps*. It becomes harder to visualize adjacent minterms for selecting PIs. You can extend the problem to six variables by using four K-Maps.



- **5 and 6-variables K Map**



Don't Care Condition

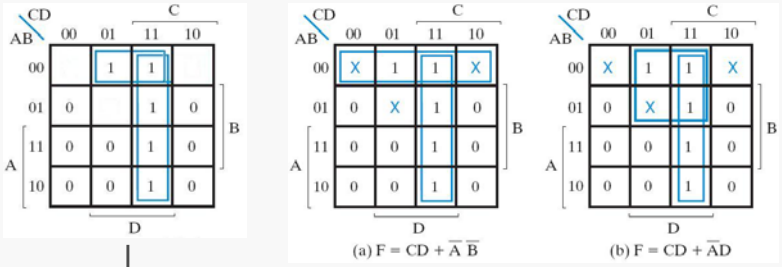
- Sometimes a function table or map contains entries for which it is known:
  - **the input values for the minterm will never occur, or**
  - **the output value for the minterm is not used**
- In these cases, the output value need not be defined
- Instead, the output value is defined as a "don't care"
- By placing "don't cares" (an "**X**" entry) in the function table or map, the cost of the logic circuit may be lowered.

- "**X**" inside a square in the map indicates that we do not care whether the value of 0 or 1 is assigned to the function for the particular minterm. **(F에 0 또는 1의 값 중 어떤 것으로 설정되든 상관하지 않는다)**
- In choosing adjacent squares to simplify the function in a map, the don't care minterms may be used. **(맵의 합수를 간략화하기 위해 인접한 사각형을 선택하는데 있어, don't care minterm이 사용될 수 있다)**

- Page 88~89, Example 2-14

  F(A,B,C,D) = ∑ *m*(1,3,7,11,15) ; variable combinations that make the function equal to 1

  ***d*(A,B,C,D) = ∑ *m*(0,2,5)** ; don't care minterms



0001 can be combine with sequare 0011 → to give a three-leteral term

- Example 1: A logic function having the binary codes for the BCD digits as its inputs. Only the codes for 0 through 9 are used. The six codes, 1010 through 1111 never occur, so the output values for these codes are "x" to represent "don't cares."

Don't Cares in K-Maps

- Example 2: A circuit that represents a very common situation that occurs in computer design has two distinct sets of input variables:
  - **A, B, and C which take on all possible combinations, and**
  - **Y which takes on values 0 or 1.**

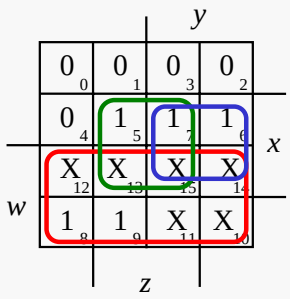
  and a single output Z. The circuit that receives the output Z observes it only for combinations of A, B, and C such A = 1 and B = 1 or C = 0, otherwise ignoring it. Thus, Z is specified only for those combinations, and for all other combinations of A, B, and C, Z is a don't care. Specifically, Z must be specified for AB + C = 1, and is a don't care for :

$$\overline{AB + \bar{C}} = (\bar{A} + \bar{B})C = \bar{A}C + \bar{B}C = 1$$

- Ultimately, each don't care "x" entry may take on either a 0 or 1 value in resulting solutions
- For example, an "x" may take on value "0" in an SOP solution and value "1" in a POS solution, or vice-versa.
- Any minterm with value "x" need not be covered by a prime implicant.

Example: BCD "5 or More"

- The map below gives a function F1(w,x,y,z) which is defined as "5 or more" over BCD inputs. With the don't cares used for the 6 non-BCD combinations:



F1(w,x,y,z) = w + x z + x y **G = 7**

- **This is much lower in cost than F2 where the "don't cares" were treated as "0s."**

F2(w,x,y,z) = $\bar{w}$ x z + $\bar{w}$ x y + w $\bar{x}$ $\bar{y}$

G = 12

- **For this particular function, cost G for the POS solution for F_1(w,x,y,z) is not changed by using the don't cares.**

Product of Sums Example

- Find the optimum POS solution:

$$F(A,B,C,D) = \Sigma_m(3,9,11,12,13,14,15) + \Sigma d(1,4,6)$$

  - Hint: Use $\bar{F}$ and complement it to get the result.

Optimization Algorithm

- Find <u>all</u> prime implicants.
- Include <u>all</u> essential prime implicants in the solution
- Select a minimum cost set of non-essential prime implicants to cover all minterms not yet covered:
  - Obtaining an optimum solution: See Reading Supplement - More on Optimization
  - Obtaining a good simplified solution: Use the Selection Rule

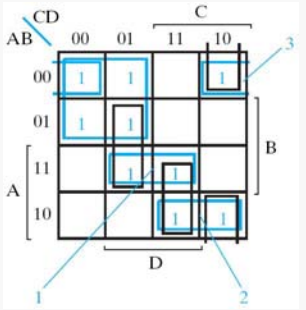
Prime Implicant Selection Rule

- Minimize the overlap among prime implicants as much as possible. In particular, in the final solution, make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.

  (주항 사이의 중복을 가능한 한 최소화한다. 마지막 답에서 적어도 하나의 최소항을 포함하도록 선택된 각 주항이 다른 주항에 포함되지 않게 해야 한다)

- Page 85~86, Example 2-12

  F(A,B,C,D) = ∑ *m*(0,1,2,4,5,10,11,13,15) ;



F(A,B,C,D) = ??

* 검정색으로 표시된 PI는 사용되지 않음.